A Project Report

on

An Intelligent TLDR Software for Summarization

Submitted to

The Department of Information Technology

In partial fulfillment of the academic requirements of
Jawaharlal Nehru Technological University
For

The award of the degree of

Bachelor of Technology
in
Information Technology
(2018-2022)

By

Sharanya Akkenapally	18311A1202
Mahima Chowdary Maddineni	18311A1226
Rokkam Krishna Vamsi	18311A1246

Under the Guidance of
Dr. Vijayalakshmi Kakulapati
Professor,
Department of IT

Sreenidhi Institute of Science and Technology

(An Autonomous Institution)

Yamnampet, Ghatkesar, R.R. District, Hyderabad - 501301

Sreenidhi Institute of Science and Technology

The Department of Information Technology

CERTIFICATE

This is to certify that this Project-1 report on "**An Intelligent TLDR Software for Summarization"** submitted by Sharanya Akkenapally(18311A1202), Mahima Chowdary Maddineni(18311A1226), and Rokkam Krishna Vamsi(18311A1246) in the year 2022 in partial fulfillment of the academic requirements of Jawaharlal Nehru Technological University for the award of the degree of Bachelor of Technology in Information Technology, is a bonafide work that has been carried out by them as part of their **Project during Fourth Year Second Semester**, under our guidance. This report has not been submitted to any other institute or university for the award of any degree.

Internal Guide	**Project Coordinator**	**Head of Department**
Dr. Vijayalaksmi Kakulapati	Dr. Vijayalaksmi Kakulapati	Dr. Sunil Bhutada
Professor	Professor	Professor
Department of IT	Department of IT	Department of IT

External Examiner
Date:

DECLARATION

We, **SHARANYA AKKENAPALLY (18311A1202), MAHIMA CHOWDARY MADDINENI (18311A1226), AND ROKKAM KRISHNA VAMSI (18311A1246)** students of **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY, YAMNAMPET, GHATKESAR,** studying IV year I semester in **INFORMATION TECHNOLOGY** solemnly declare that the Project-1 work, titled "**An Intelligent TLDR Software for Summarization**" is submitted to **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY** for partial fulfillment for the award of the degree of Bachelor of technology in **INFORMATION TECHNOLOGY**.

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for the award of any degree.

Sharanya Akkenapally-18311A1202
Mahima Chowdary Maddineni-18311A1226
Rokkam Krishna Vamsi-18311A1246

ACKNOWLEDGEMENT

We would like to express my gratitude to all the people behind the screen who helped me to transform an idea into a real application.

We would like to express my heartfelt gratitude to my parents without whom I would not have been privileged to achieve and fulfill my dreams. I am grateful to our CEO **Mr. Abhijit Rao Katikaneni** , director **Dr. C.V. Tomy**, principal **Dr. T. Ch. Siva Reddy** and management**,** who most ably runs the institution and has had a major hand in enabling me to do my project.

We profoundly thank **Dr. Sunil Bhutada,** Head of the Department of Information Technology who has been an excellent guide and also a great source of inspiration for my work.

We would like to thank our Internal Guide and Project Coordinator **Dr. Vijayalakshmi Kakulapati,** Professor, for her technical guidance, constant encouragement, and support in carrying out my project at college.

The satisfaction and euphoria accompanying the successful completion of the task would be great but incomplete without the mention of the people who made it possible with their constant guidance and encouragement crowns all the efforts with success. In this context, I would like to thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

Sharanya Akkenapally-18311A1202
Mahima Chowdary Maddineni-18311A1226
Rokkam Krishna Vamsi-18311A1246

Abstract

In the significant data era, there has been an explosion in the amount of text data from various sources. This text volume is an invaluable source of information and knowledge that needs to be effectively summarised to be valid. It requires ample time to go through and keep up with all the data present, as billions of articles are generated every day. There is a need to reduce much of this text data to shorter, focused summaries that capture the salient details, both so we can navigate it more effectively and check whether the larger documents contain the information we are looking for. Since manual text summarization is a time-expensive and generally uphill task, the automatization of the task is gaining increasing fame and therefore constitutes an excellent motivation for academic research. This increasing availability of documents has demanded exhaustive research in the NLP area for automatic text summarization. The real question is, "Is there any software that can help us absorb the data more effectively and in less time?" So, The main objective of the summarization system is to identify the most critical information from the given and present it to the end-users. Summarization in NLP is the process of summarising the text information in large texts for easy interpretation and quicker consumption. We propose a solution by implementing an application for text summarization using Natural Language Processing by accepting an input (plain text or text scrapped from a website). The outlined text is given as output. Natural language, along with machine learning processing, made it simpler to summarise lengthy amounts of text into a cohesive and fluent summary that only includes the article's essential ideas.

1. Introduction

There is a huge quantity of written material, and each and every day, that quantity is simply going to increase further.

Imagine the internet, which is made up of different web pages, items of news, status updates, blogs, and a great deal more. Because the data are not organised in any particular way, the only way to traverse them is to perform a search and then scan through the results.

There is a significant requirement for reducing a significant portion of this text data to shorter, more focused summaries that capture the essential details. This is necessary not only to enable us to navigate it with greater efficiency but also to determine whether or not the more extensive documents contain the information that we are looking for.

1.1 Need and Objective

It requires ample time to go through and keep up with all the data present, as billions of articles are generated every day. There is a need to reduce much of this text data to shorter, focused summaries that capture the salient details, both so we can navigate it more effectively and check whether the larger documents contain the information we are looking for. Since manual text summarization is a time-expensive and generally uphill task, the automatization of the task is gaining increasing fame and therefore constitutes an excellent motivation for academic research. This increasing availability of documents has demanded exhaustive research in the NLP area for automatic text summarization.

1.2 Scope of Research

Among various types of data generated, texts are the most general over the network. The information overload problem worsens as the quantity of data keeps increasing rapidly. Text summarization contracts input text into a concise and fluent summary conveying the critical information in the original text. Text summarization approaches can be of two types: extractive summarization and abstractive summarization. Extractive summarization generates the summary from the sentences in the input, and abstractive summarization creates the generalization of the input using different words.

2. Literature Survey

2.1 Problem statement

Textual data in digital documents quickly adds up to massive amounts of information. The majority of this enormous amount of unstructured data is unconstrained and hasn't been arranged into typical databases.

As a result, processing documents is a purely administrative task mostly as a result of a lack of norms. As a result, there is a requirement for a system to reduce the text's size, give it some structure, and process the document to make it more readable. It is understandable to the user.

This project aims to address all the above aspects of the problem. With extractive text summarization, the world's expanding amount of textual information

2.2 Proposed System

There are now automatic text summarizers available, each of which uses a unique methodology. The extraction of statistical information based on the frequency and distribution of words is a common strategy. Selecting phrases with the highest scores from those obtained is a common next step. Another method uses this information in conjunction with the positioning of sentences inside the paragraph, cue words, and title and heading terms that are included in the paragraph. The method used in this project uses a Text Rank technique, which analyses a passage based on a number of important characteristics to evaluate whether or not a certain sentence ought to be included in the summary. If one were to tokenize the text into paragraphs and sentences and then analyze each sentence individually, one could be able to construct a thorough summary of the content.

3. System Analysis

3.1 Software requirements

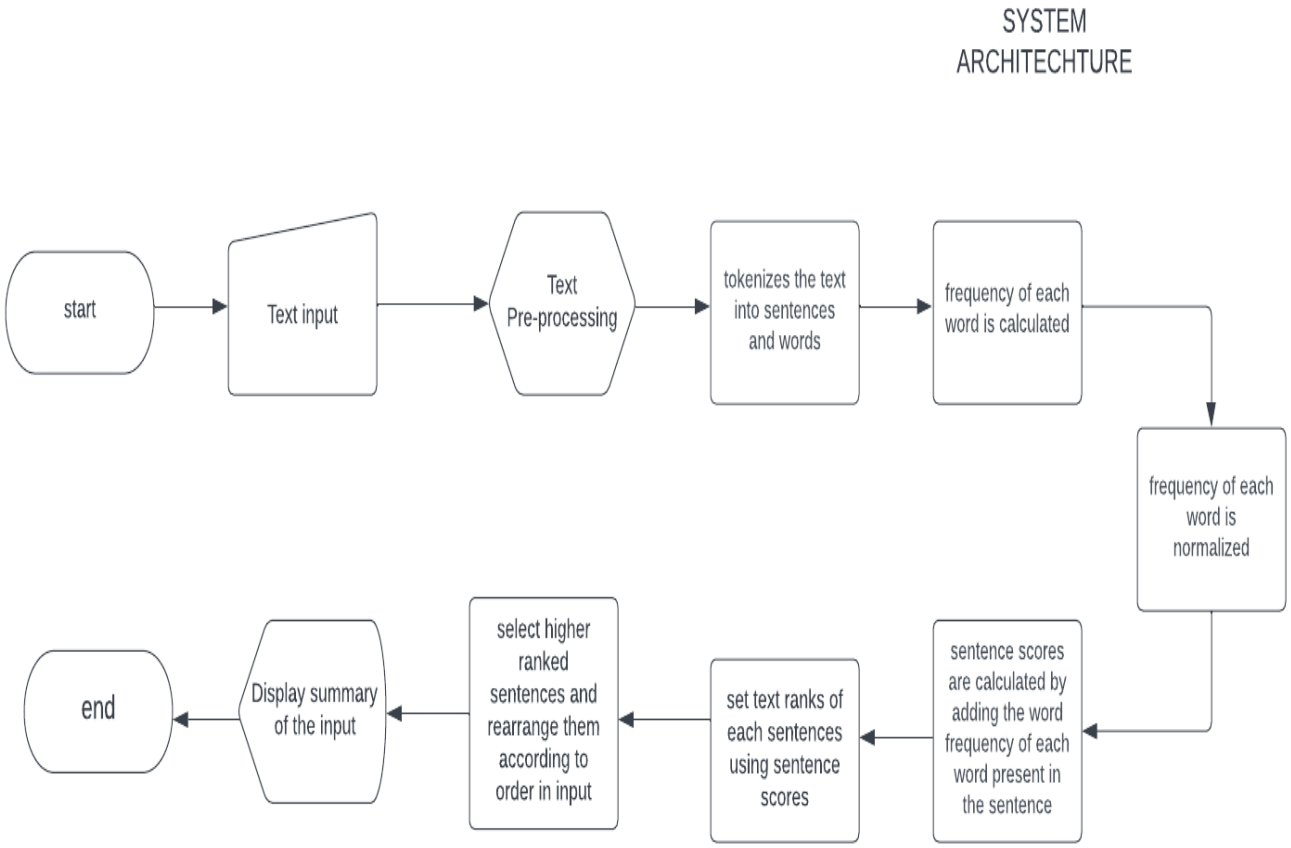
1. Python version 3.7or above
2. Required: Spacy, rouge, heapq, bs4, urllib, re

3.2 Hardware requirements

1. Processor**:** i5 processor
2. **RAM**: 4GB
3. **Hard Disk**: 20GB

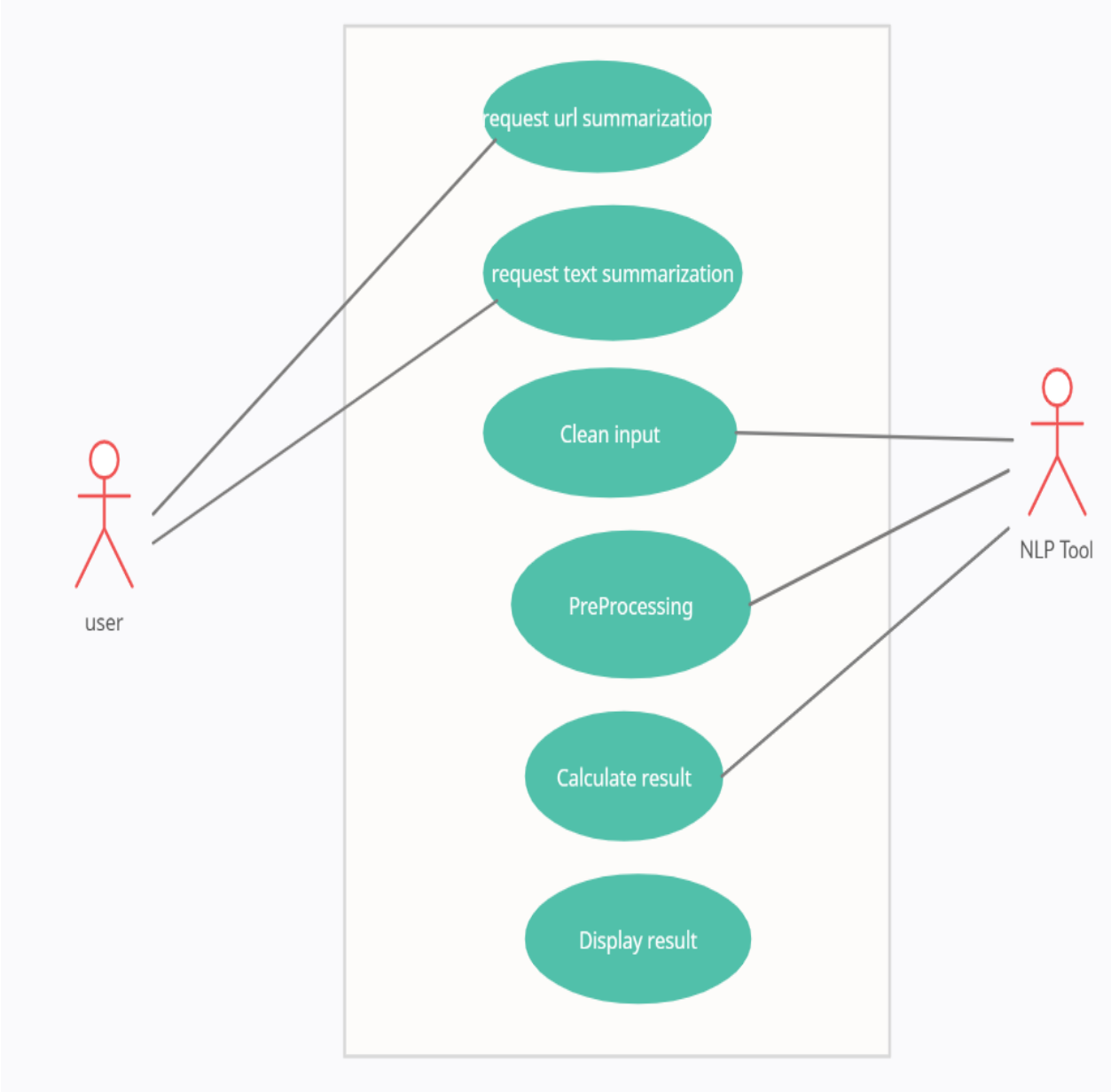
4. System Design

4.1 Architecture of proposed system

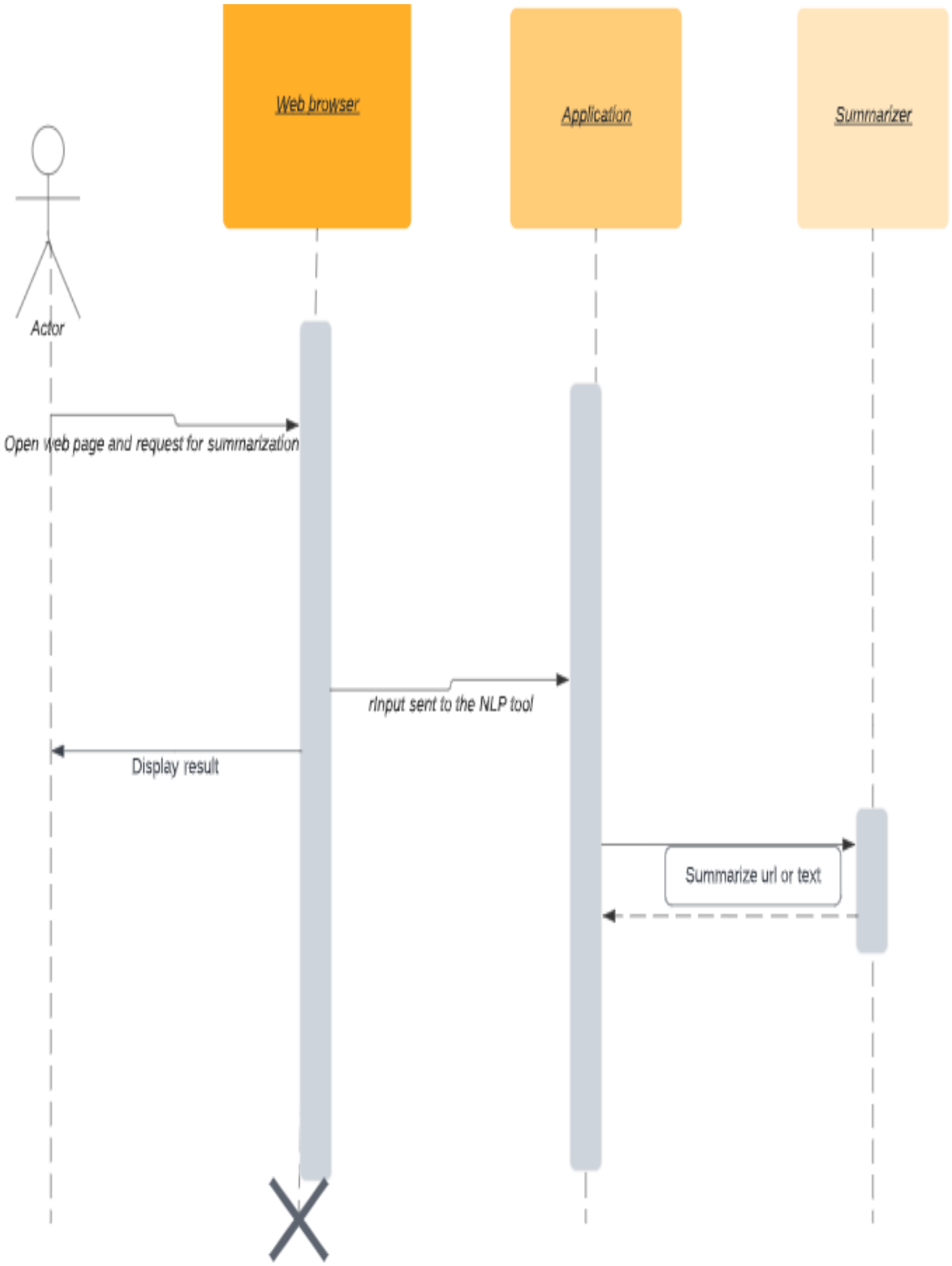


4.2 UML Diagrams

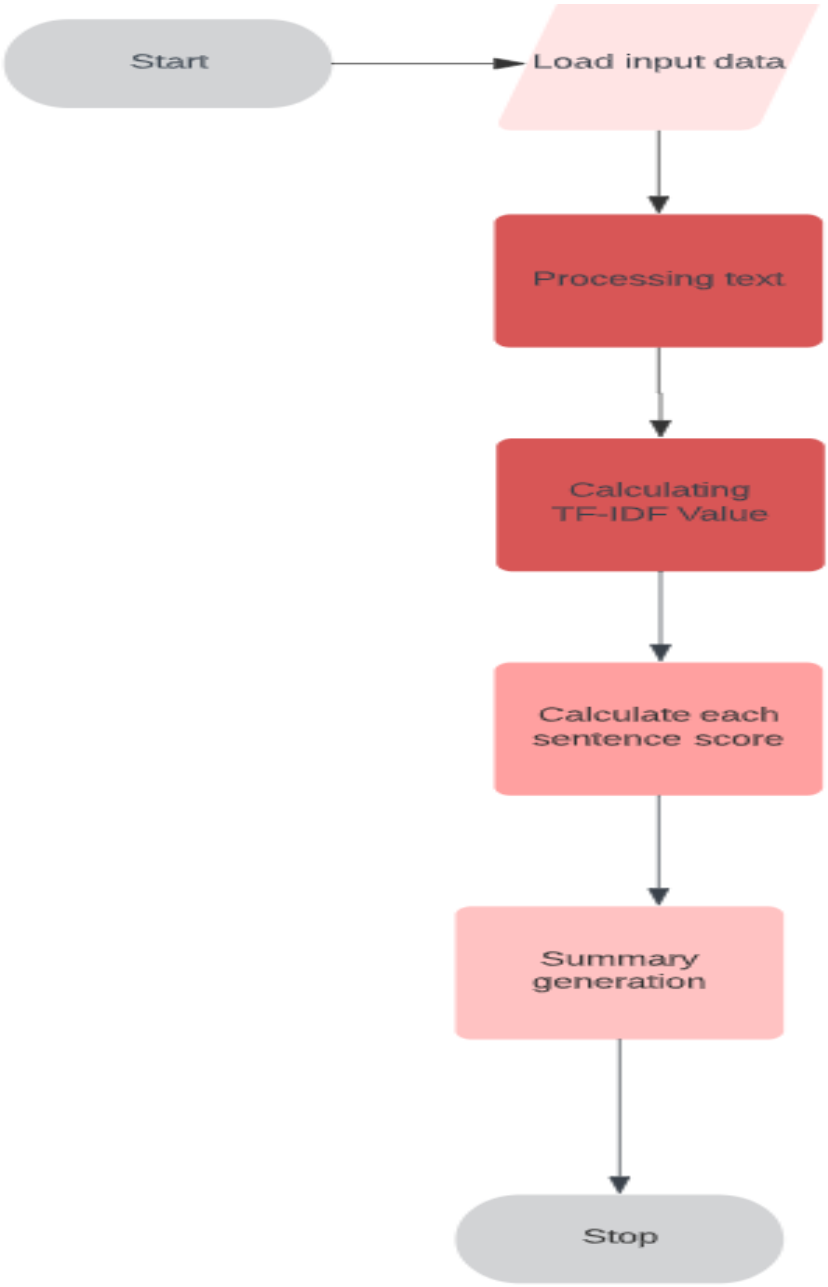
Uml use case diagram



Sequence uml diagram:



Activity uml diagram



4.3 Modules:

- Pre-processing module:The data are prepared for analysis by the pre-processing module, which consists of pre-processing procedures. In order to get rid of the effects that the detectors have on the data, we need to perform some preliminary processing on it first. The phase of data processing known as preprocessing is considered to be the most crucial. After an experiment has been completed and data has been collected as an output, the following step is to model the data in order to derive knowledge that is of use. The output of data on a global scale may either be substantial, insufficient, or fragmented. The initial step in the processing of data is known as "pre-processing," and it involves sorting the data into one of these three categories before further processing. In light of this, several data preparation tasks, such as data filtering, data ordering, data editing, and noise modelling, all play a significant part.

- Web scrapping module: Web scraping is used to extract data from websites. Web scraping software may use HTTP or a web browser to access the Web. Specific web data is captured and copied into a local database or spreadsheet for later retrieval or analysis. Web programming uses Python heavily. When browsing websites, we use the web address, or URL. Python's built-in materials can handle URL calls and pass the result. This article discusses urllib. We'll also look at the module's functions for getting URL results.

- NLP module: spaCy is a Python and Cython open-source NLP package. Matthew Honnibal and Ines Montani, founders of Explosion, developed the MIT-licensed library. spaCy focuses on producing software for production, unlike NLTK. spaCy provides deep learning processes that connect statistical models built by TensorFlow, PyTorch, or MXNet with its own library Thinc. Thinc's backend, spaCy, uses convolutional neural network models for part-of-speech tagging, dependency parsing, text categorization, and named entity recognition (NER). There includes a multi-language NER model and prebuilt statistical neural network models for 17 languages, including English, Portuguese, Spanish, Russian, and Chinese. Users can train bespoke models on their own datasets with tokenization support for 65 languages.

5. Implementation:

5.1 Language/Technology used for implementation

Natural language processing (NLP) focuses on making algorithms to understand natural human language. Natural Language Toolkit, or NLTK, is a Python package that can be used for NLP. Python has numerous packages that can be used to reuse code. It has transparent semantics and syntax, making it an excellent choice for natural language processing. It's also straightforward to use and has excellent support for integrating with other tools and languages. The most significant advantage of Python for natural language processing is that it provides developers with a large number of libraries that handle a variety of NLP-related tasks such as topic modelling, document categorization, sentiment analysis, and so on. It is undeniably tough to create software that can deal with natural language. Python's broad toolset, on the other hand, allows developers to create fantastic tools.

5.2 Algorithms Implemented

Text rank

TextRank is an algorithm that is frequently used in keyword extraction and text summarising. It is based on the PageRank algorithm. In order to determine which sentences in a body of text are the most pertinent, a graph is constructed. The vertices of the graph represent each sentence in the document, and the edges between sentences are determined by the amount of content overlap between the sentences; specifically, this is done by counting the number of words that both sentences share.

The Pagerank algorithm, which determines which sentences are the most essential based on their weight in the network of sentences, is fed the sentences that make up this network. Now, when we wish to extract a summary of the text, we can take only the sentences that are the most important.

The textrank algorithm creates a word network in order to locate meaningful keywords in the content. This network is formed by looking which words follow one another. A connection is made between two words if they come one after the other in the text; this connection is given more significance if the two words in question appear in close proximity to one another more frequently.

The Pagerank algorithm is implemented on top of the generated network in order to determine the significance of each individual word. Only the first one third of all of these words are retained since they are regarded to be significant. Following this step, a keywords table is created by grouping

together the pertinent terms in the text in the event that they are found immediately following one another.

```
1.  read text
2.  for each line in text do
3.    tokenize
4.    stop words removal
5.  end for
6.  for each word in tokens list
7.    count frequency
8.  end for
9.  normalize frequencies to generate word score
10. add word scores to find sentence scores
11. sort sentences by score
12. determine desired length
13. while(desired length is not met)
14.   for( all sentences x)
15.         if(sentence x not already in summary)
16.               set summary_sentence += x
17.         end if
18.   end for
19. end while
20. output summary
```

5.3 Code

You may occasionally receive or copy data that contains superscripts or subscripts that you do not wish to be included. If you've imported or copied data into your text that contains superscripts or subscripts, you should remove them before doing any additional analysis or processing. This can be a labor-intensive and time-consuming manual task. The following function is used to remove superscript citations and square brackets from text.

```
import bs4 as bs
import urllib.request
import re

def remove_brackets(text):
        '''Removes the citations and square brackets in super script from text'''
        text = re.sub(r'\[[0-9]*\]', ' ', text)
        return re.sub(r'\s+', ' ', text)
```

Web scraping refers to the practice of extraction of data from websites through the use of an automated procedure. Web scraping may be a tiresome process, regardless of the programming language you use. The structure of no two websites is precisely the same, and HTML is frequently disorganized. In addition, websites undergo ongoing revisions. It is not assured that web scrapers that operate now will continue to work one year from now – or even one week from now! The urllib package in Python's standard library is handy for web scraping because it offers tools for interacting with URLs. This package may be found in the standard library. Specifically, the urllib.request module includes a function known as urlopen() that can be used to open a URL within a program. This function can be found in the urllib.request module. Regular expressions, often known as regexes for short, are a type of pattern that can be applied to a string in order to look for particular text. Regular expressions may be executed in Python thanks to the re module found in the standard library. In order to distinguish between various patterns, regular expressions make use of specialised characters referred to as metacharacters. For example, the character represented by an asterisk (*) denotes the presence of zero or more of whatever comes immediately before the asterisk. The.find() method would have a tough time dealing with the inconsistencies seen here, but with some ingenuity and the usage of regular expressions, you can manage this code in a rapid and effective manner.

```
def scraper(url):
    '''Scrapes content from paragraph and div tags of website'''
    article_text = ""

    user_agent = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7) Gecko/2009021910 Firefox/3.0.7'
    headers={'User-Agent':user_agent,}

    request=urllib.request.Request(url,None,headers)
    response = urllib.request.urlopen(request)
    data = response.read()

    parsed_article = bs.BeautifulSoup(data, 'lxml')

    title = parsed_article.find('title').text

    paragraphs = parsed_article.find_all('p')

    # Checking content of p tags
    if len(paragraphs) != 0:
        for p in paragraphs:
            article_text += p.text

    # Checking content of div tags
    else:
        divs = parsed_article.find_all('div', id = 'container')

        for div in divs:
            article_text += div.text

    body = remove_brackets(article_text)

    return title, body
```

An abbreviation for "words per minute" is usually abbreviated WPM, which measures the number of words typed or read in a minute. WPM has a wide range of meanings and nuances. We can't objectively determine an average reading time. Second, the length or duration of words is clearly changeable, as certain words (like "dog") can be read relatively fast. In contrast, others (like "rhinoceros") take a significantly longer period of time to complete the reading. As a result, the usual definition for most words is five letters long. Font size and type, your age, whether you're reading on a monitor or paper, and even the number of paragraphs, images, and buttons in the article's website all affect how long it takes to read an

Words will be defined as five characters (including spaces and punctuation), and WPM will be set to 200 to make things easier to understand.

```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
# Used to rank sentences according sentence scores
from heapq import nlargest

def estimated_reading_time(text):
    '''Calculating reading speed by dividing
    text length by average reading speed (avg words per pm)'''
    mins = int(len(text)/200)
    seconds = int((float(len(text)/200) - mins)*60)
    return "( Estimated reading time: {} mins, {} seconds )".format(str(mins),str(seconds))
```

To get started, you need to import spaCy and any other required modules. Importing spaCy and any other prerequisite modules is the first step. In the NLP function, you should pass in the string doc. The len function is used in the following example to determine the total number of sentences included in the string. Following this, two lists will be compiled for parts of speech and stop words in order to validate each token. After this, the necessary tokens will be filtered out, and the validated tokens will be saved in the tokens list. Utilizing the "Counter" function, compute the frequency of each token and then save the results in the word frequencies variable. The words may be the same, but they could be in upper or lower case. In order to facilitate a quick and accurate assessment, all of the words were capitalized.

```
def summarizer(text):
    '''Summarizes text by tokenizing, creating a word frequency list,
       finding sentence scores, and then selecting sentences with
       highest sentence scores'''

    stopwords = list(STOP_WORDS)
    #print(stopwords)

    # Loading model for tokenization
    nlp = spacy.load('en_core_web_sm')

    # Tokenizing text with spacy
    doc = nlp(text)

    tokens = [token.text for token in doc]
    #print(tokens)

    # Finding Word Frequencies
    word_frequencies = {}

    for word in doc:
        if word.text.lower() not in stopwords:
            if word.text.lower() not in punctuation:
                if word.text.lower() not in word_frequencies.keys():
                    # Adding new word to word_frequency
                    word_frequencies[word.text.lower()] = 1
                else:
                    # Incrementing frequency in word already exists
                    word_frequencies[word.text.lower()] += 1
```

This frequency can be normalized for more efficient processing by dividing the token's frequencies by the greatest frequency. Normalization can be accomplished in this manner. This is the most important step, in which the value of each sentence is determined by the percentage of times the token appears in that sentence. The outcome is kept as a key-value pair in the sentence scores variable, with the sentences from the string doc serving as the keys and the weights assigned to each sentence serving as the values.

```
# Normalizing Word Frequencies
max_frequency = max(word_frequencies.values())
#print(max_frequency)

for word in word_frequencies.keys():
    word_frequencies[word] /= max_frequency

#print(word_frequencies)

# Sentence Tokenization
sentence_tokens = [sent for sent in doc.sents]
#print(sentence_tokens)

# Calculating sentence scores
sentence_scores = {}

for sent in sentence_tokens:
    for word in sent:
        if word.text.lower() in word_frequencies.keys():
            if sent not in sentence_scores.keys():
                sentence_scores[sent] = word_frequencies[word.text.lower()]
            else:
                sentence_scores[sent] += word_frequencies[word.text.lower()]
```

Last but not least, the nlargest function is used to summarise the string. This function requires three arguments: the number of data to extract, an iterable (list, tuple, or dictionary), and the condition to be satisfied. The top twenty percent of sentences in terms of the score are chosen to serve as the selected sentences parameter for the nlargest function. It returns a list that contains the sentences that have been saved under the name summary_sentences.

```
# Getting Sentences with highest scores
sentences_percent = 0.2
sentences_selected = int(len(sentence_tokens)*sentences_percent)
#print(sentences_selected)

#heapq.nlargest(selectCount, iterable, keys )
summary_sentences = nlargest(sentences_selected, sentence_scores, key = sentence_scores.get)
#print(summary_sentences)
summary_sentences = [word.text for word in summary_sentences]
summary = " ".join(summary_sentences)
return summary
```

Python is the language that was used to write the Flask micro web framework. It is referred regarded as a microframework due to the fact that it does not need any specific tools or libraries to function. It does not have a DB(database) abstraction layer, a form validation layer, or any other

components that would normally be provided by pre-existing third-party libraries for the same duties. Flask, on the other hand, has support for extensions that let developers add functionality to their applications just as if the functionality had been built into Flask itself. There are extensions available for object-relational mappers, form validation, upload handling, a variety of open authentication protocols, and other utilities connected to standard frameworks.

```
from flask import Flask, render_template, request, url_for
from scraper import scraper
from summarizer import summarizer, estimated_reading_time

app = Flask(__name__)
app.config["JSON_AS_ASCII"] = False

@app.route("/")
def home():
        return render_template("main.html")

@app.route("/text", methods=['GET', 'POST'])
def text():
    if request.method == 'POST':
        text = request.form.get('text')
        summary = summarizer(text)
        reading_time = estimated_reading_time(summary.split())
        return render_template("text.html",reading_time = reading_time, summary = summary)

    else:
        return render_template("text.html")
```

```
@app.route("/index", methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        url = request.form.get('url')
        try:
            article_title, text = scraper(url)
            summary = summarizer(text)
            reading_time = estimated_reading_time(summary.split())
            return render_template("index.html", article_title = article_title, reading_time = reading_time, summary = summary)

        except TypeError():
            print('Invalid url entered')

    else:
        return render_template("index.html")

if __name__ == "__main__":
    app.run(debug = True)
```

Html code

main.html

```
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset='utf-8'>
    <meta name='viewport' content='width=device-width'>
    <meta name='description' content='Text Summarization Web Application'>
    <link rel='stylesheet' href="{{ url_for('static', filename = 'css/main.css') }}" }>
    <title>Text Summarizer</title>
</head>

<body>
      <label class='heading' id='heading'>Text Summarizer</label>
    <div class='container'>
      <p>Summarization is the task of condensing a piece of text to a shorter version,
reducing the size of the initial text while at the same time preserving key informational elements
 and the meaning of content. Since manual text summarization is a time expensive and generally laborious task,
 the automatization of the task is gaining increasing popularity and therefore constitutes a strong motivation for academic research.
In the big data era, there has been an explosion in the amount of text data from a variety of sources.
This volume of text is an inestimable source of information and knowledge which needs to be effectively summarised to be useful.
 This increasing availability of documents has demanded exhaustive research in the NLP area for automatic text summarization.
So, The main objective of a text summarization system is to identify the most important information from the given text
and present it to the end users. Text summarization in NLP is the process of summarising the text information in large texts
 for easy interpretation and for quicker consumption . In this project, we will walk you through the traditional extractive
as well as the advanced generative methods we used to implement this functionality using NLP and Machine learning techniques in python.
 So we aim at developing an AI/ML based solution that generates short summaries of press releases for easy and more targeted
dissemination of information to the people. Such a TL,DR (Too Long, Didn't Read) software may be integrated in an app-based solution.  </p>
    <br> <br> <br>
        <a href="index"><button name='url' class='summarize_btn'>url</button></a>
  <br> <br>
        <a href="text"><button name='text' class='summarize_btn'>text</button></a>
    </div>
</body>
</html>
```

Index.html

```
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset='utf-8'>
    <meta name='viewport' content='width=device-width'>
    <meta name='description' content='Text Summarization Web Application'>
    <link rel='stylesheet' href="{{ url_for('static', filename = 'css/main.css') }}" }>
    <title>Text Summarizer</title>
</head>

<body>

    <label class='heading' id='heading'>Text Summarizer</label>
    <div class='container'>
        <label class='paste_label'>Paste url of article below</label> <br>
        <form action='#' method='POST'>
            <input type='url' name='url' autocomplete='off' placeholder='https://en.wikipedia.org/wiki/Internet'> <br>
            <button name='summarize_btn' class='summarize-button'>Summarize</button>
        </form> <br>
        <label class='summary_label'>Summary</label> <br>
        <label class='article_title'> {{ article_title }} {{reading_time}} </label> <br>
        <textarea> {{ summary }} </textarea>
    </div>
</body>
</html>
```

Text.html

```
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset='utf-8'>
    <meta name='viewport' content='width=device-width'>
    <meta name='description' content='Text Summarization Web Application'>
    <link rel='stylesheet' href="{{ url_for('static', filename = 'css/main.css') }}" }>
    <title>Text Summarizer</title>
</head>

<body>

    <label class='heading' id='heading'>Text Summarizer</label>
    <div class='container'>
        <label class='paste_label'>Paste text below</label> <br>
        <form action='#' method='POST'>
            <input type='textarea' name='text' autocomplete='off' placeholder='paste your text'> <br>
            <button name='summarize_btn' class='summarize_btn'>Summarize</button>
        </form> <br>
        <label class='summary_label'>Summary</label> <br>
        <label class='article_title'> {{reading_time}} </label> <br>
        <textarea> {{ summary }} </textarea>
    </div>
</body>
</html>
```

Css file

```
body{
    margin: 0px 0px;
    font-family: Times New Roman;
    background-color: whitesmoke;
    color: black;
}

.container
{
    text-align: center;
    padding: 30px;
}

label{
   font-family: "Comic Sans MS", "Comic Sans", cursive;
}

label.paste_label{
    size: 2em;
    width: 100%;
    text-align: center;
    padding: 7px 0px;
    font-weight: bolder;
    font-size: 3.5em;
}

label.heading{
  font-family: 'Brush Script MT', cursive;
    padding-left: 10px;
    font-size: 2em;
    font-weight: bold;
background: -webkit-linear-gradient(#00FF00, #008080);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;

}
```

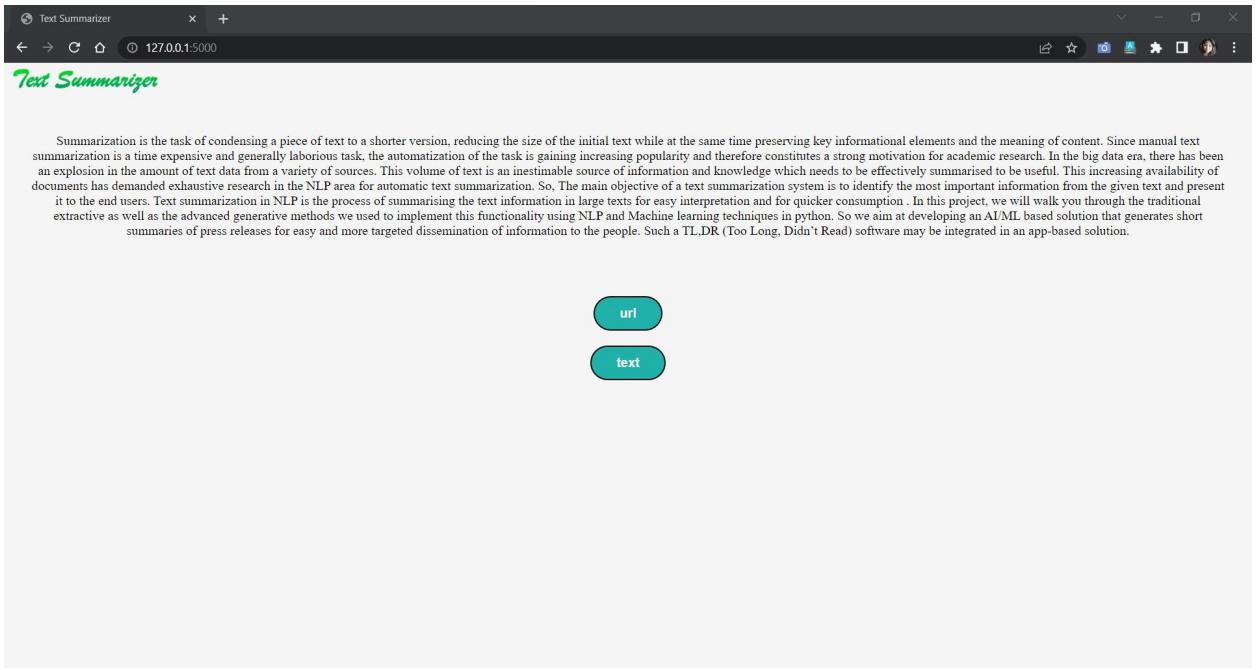
```
input{
    left: 10px;
    width: 450px;
    margin: 25px 0;
    padding: 10px 10px;
    border:2px solid black;
    border-radius: 30px;
    background-color: C0C0C0;
    color: black;
    outline: none;
}

button{
    left: 15px;
    padding: 10px;
    border-radius: 30px;
    padding: 10px 30px;
    font-weight: bold;
    font-size: 1em;
    color: ivory;
    background-color:LightSeaGreen;
    outline: none;
}

button:hover{
    color:white;
    transform: scale(1.05);
}

button:active{
    transform: translateY(2px);
}
```

6. Results:

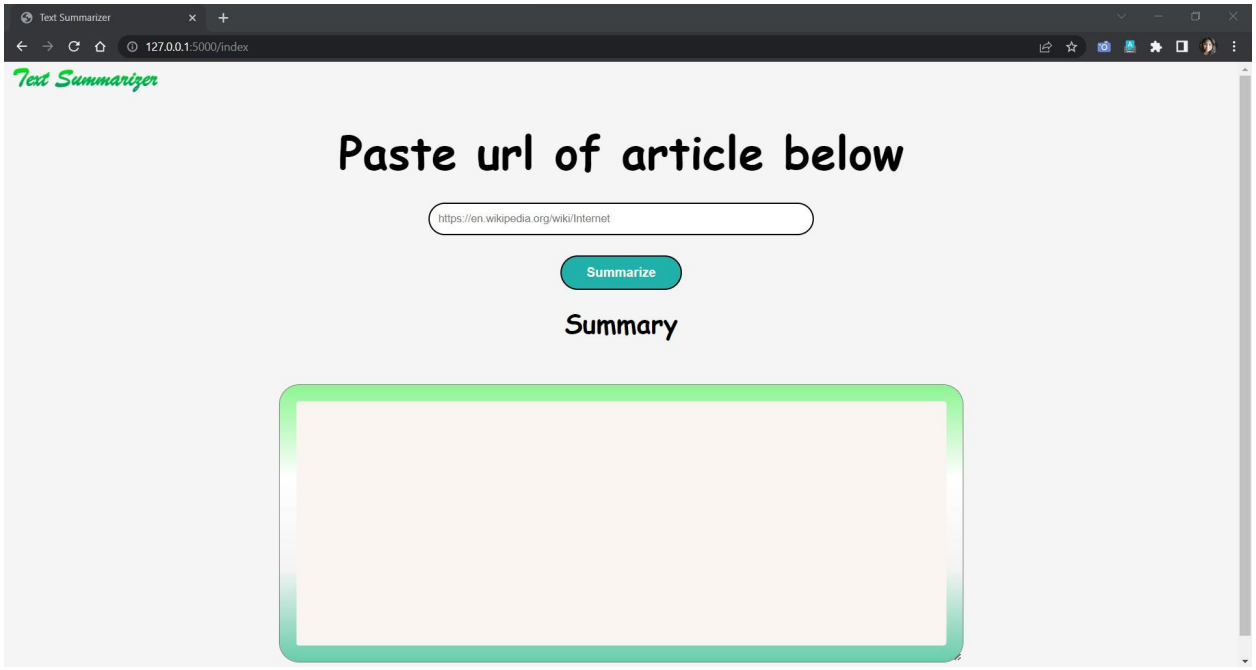
6.1 Outputs

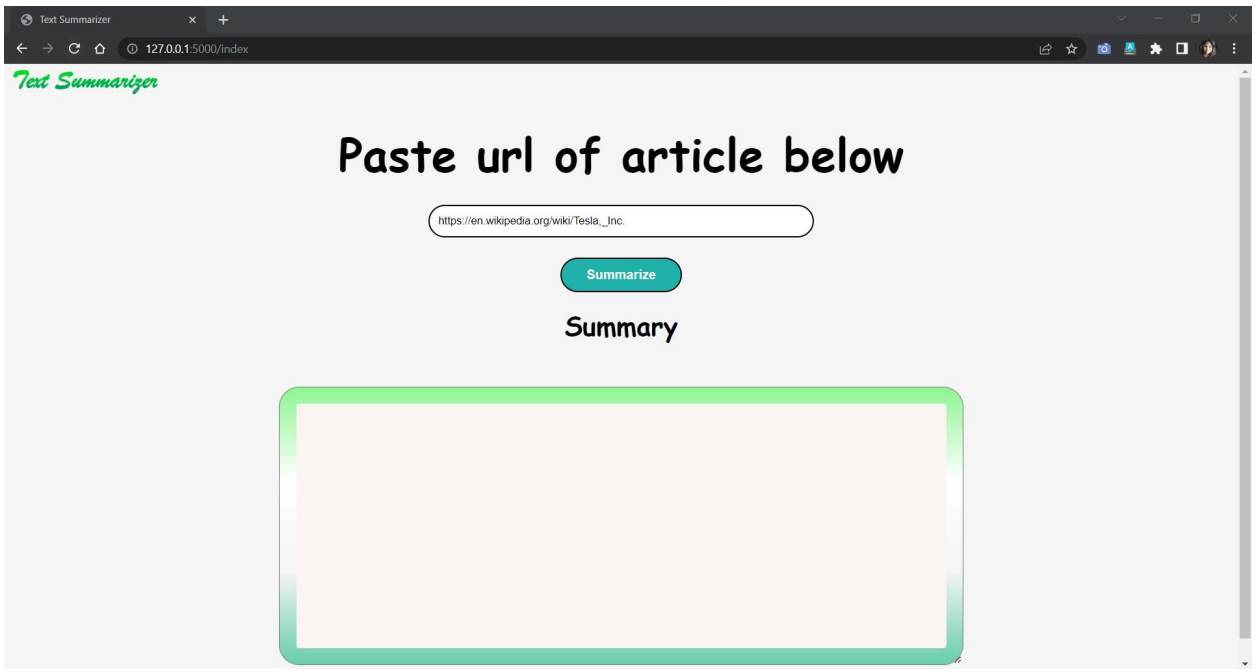
This is an extractive text summarizer written with Python and Flask. It accepts a URL as input, generates a summary, and then provides an estimate of the amount of time it will take to read the summary.

- Run app.py in anaconda prompt
- Simply follow the link that is provided on the screen of the terminal. In most cases, the local host:5000
- choose which website or text you would want to input.

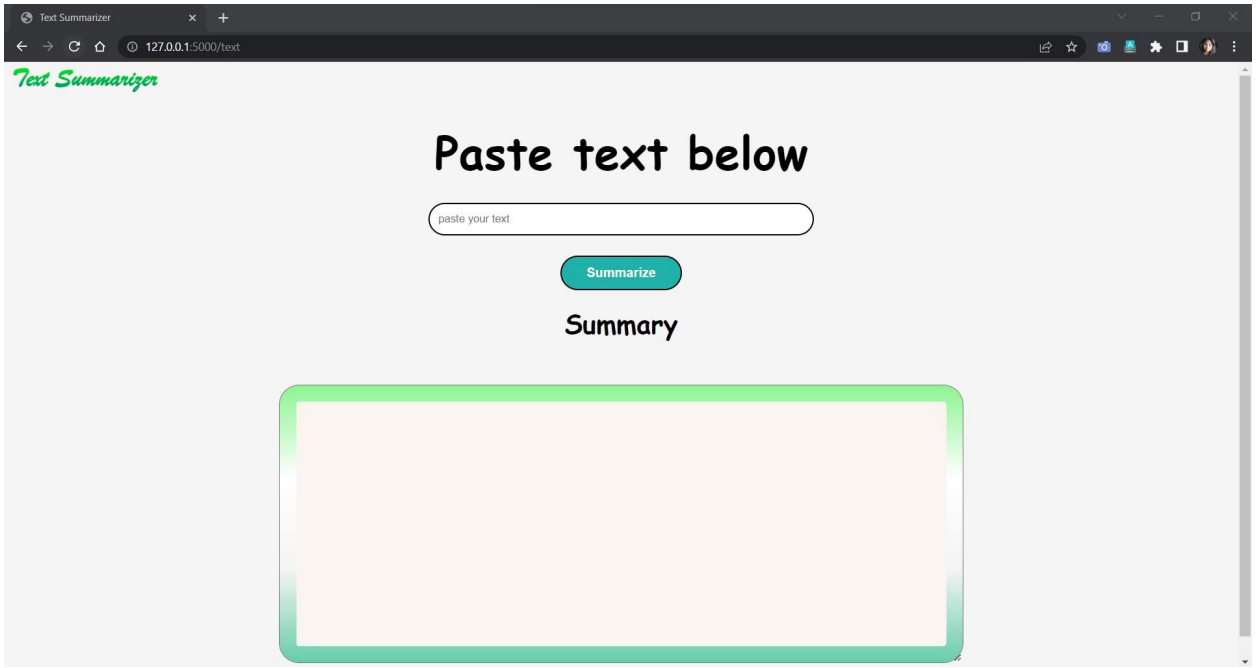


- in the event that url is chosen:
    - Copy the url and paste it into the box labelled "url."
    - To obtain a summary, click the button labelled "Summarize." After being given a URL, the scraper.py programme collects the text that is displayed on the webpage. After that, the text is cleaned up and formatted.
    - The summarizer.py programme is then given this formatted text to process. The display concludes with the book's title, a brief summary, and an approximate reading time.


Text Summarizer
127.0.0.1:5000/index
Text Summarizer
Paste url of article below
https://en.wikipedia.org/wiki/Internet
Summarize
Summary


Text Summarizer
127.0.0.1:5000/index
Text Summarizer
Paste url of article below
https://en.wikipedia.org/wiki/Tesla,_Inc.
Summarize
Summary

- If the text option is chosen
    - Copy the text that is now chosen and then paste it into the text area.
    - To obtain a summary, click the button labelled "Summarize." The text is sent to the summarizer.py script to process. At the very end, a brief summary and an estimated reading time are presented.



6.2 Result Analysis

ROUGE is an abbreviation that refers for "recall-oriented understudy for pointing evaluation." In its most basic form, it is a set of measures that may be used to evaluate the automatic summarization of texts as well as machine translations.

It accomplishes this by contrasting a translation or summary that was generated automatically with a collection of reference summaries (typically human-produced).

We can really compute the precision and recall by using the overlap, which will allow us to obtain a good quantitative result.

To put it another way, recall (in the context of ROUGE) refers to the portion of the reference summary that is being recovered or captured by the system summary. If we ignore everything but the words themselves, we can arrive at the following conclusion:

Recall = number of overlapping words / Total words in reference summary

This seems like it would make an excellent text summarising system. However, it does not present the opposing viewpoint to the argument. A system summary, which is generated by a machine, can be very lengthy because it includes all of the words that are in the reference summary. However, it's possible that many of the words in the system overview aren't necessary, which makes the summary more wordy than it needs to be.

Here is where the need for precision becomes apparent. In terms of accuracy, what you are ultimately measuring is the proportion of the system summary that was actually necessary or relevant to the purpose of the analysis. Precision can be measured as:

Precision = Number of overlapping word / Total words in system summary

When attempting to develop summaries that are condensed in nature, accuracy becomes an extremely important factor to take into consideration. Because of this, it is always recommended to first compute the precision and then the recall before reporting the F-measure.

If your summaries are in some manner compelled to be succinct as a result of some constraints, then you might want to think about using just the recall instead of the precision because accuracy is less of an issue in this circumstance.

Three different granularity levels can be used to compare the system summaries to reference summaries: ROUGE-S, ROUGE-L, or ROUGE-N

ROUGE-N — measures unigrams, bigrams, trigrams, and orders of magnitude higher n-grams that share a same root

Rouge-L measures the longest matching sequence of words using LCS. Using LCS is advantageous because it does not require consecutive matches, but rather in-sequence matches that reflect sentence-level word order. " You don't need to specify a specific n-gram length because it automatically includes the longest in-sequence common n-gram.

Doesn't matter in which order the words are in, as long as there are no gaps between them. Skip-gram concurrence is another term for this. Word pairs with a maximum of two spaces in between them can have their overlap measured using skip-bigram, for example. Let's say you want to skip-bigram "cat in the hat," the skip-bigrams for that sentence would be something like "cat hat" for example.

Using ROUGE-1 as an example, it refers to the number of unigrams that overlap between the system summary and the reference summary. Bigram overlap between the system and reference summaries is referred to as ROUGE-2.

```
txt="""Due to the global attention that it has received and the millions of visitors it attracts, the Taj Mahal has become a pror

Along with being a renowned symbol of love, the Taj Mahal is also a symbol of Shah Jahan’s wealth and power, and the fact that th

The planning and structure of the Taj Mahal, from the building itself to the gardens and beyond, is symbolic of Mumtaz Mahal’s ma

According to Ebba Koch, art historian and international expert in the understanding and interpretation of Mughal architecture and
The Taj is also seen as a feminine architectural form, and is thought to embody Mumtaz Mahal herself."""
txt
```

```
'Due to the global attention that it has received and the millions of visitors it attracts, the Taj Mahal has become a prominen
t image that is associated with India, and in this way has become a symbol of India itself.[49]\n\nAlong with being a renowned
symbol of love, the Taj Mahal is also a symbol of Shah Jahan’s wealth and power, and the fact that the empire had prospered und
er his rule.[50] Bilateral symmetry dominated by a central axis has been used by rulers as a symbol of a ruling force that brin
gs balance and harmony, and Shah Jahan applied that concept in the making of the Taj Mahal.[51] Additionally, the plan is align
ed in the cardinal north-south direction and the corners have been placed so that when seen from the center of the plan, the su
n can be seen rising and setting on the north and south corners on the summer and winter solstices respectively. This makes the
Taj a symbolic horizon.[52]\n\nThe planning and structure of the Taj Mahal, from the building itself to the gardens and beyond,
is symbolic of Mumtaz Mahal’s mansion in the garden of Paradise.[51] The concept of Gardens of Paradise is extended into the bu
ilding of the mausoleum as well. Colorful vines and flowers decorate the interior, and are filled in with semi-precious stones
using a technique called pietra dura, or as the Mughals called it, parchin kari.[53] The building appears to slightly change co
lor depending on the time of day and the weather. The sky has not only been incorporated in the design through the reflecting p
ools but also through the surface of the building itself. This is another way to imply the presence of Allah at the site.[54]\n
\nAccording to Ebba Koch, art historian and international expert in the understanding and interpretation of Mughal architecture
and the Taj Mahal, the planning of the entire compound of the Taj symbolizes earthly life and the afterlife, a subset of the sy
mbolization of the divine. The plan has been split into two—one half is the white marble mausoleum itself and the gardens, and
the other half is the red sandstone side meant for worldly markets. Only the mausoleum is white so as to represent the enlighte
nment, spirituality and faith of Mumtaz Mahal. According to the world-traveler Eleanor Roosevelt, the white symbolized the puri
ty of real love.[55] Koch has deciphered that symbolic of Islamic teachings, the plan of the worldly side is a mirror image of
the otherworldly side, and the grand gate in the middle represents the transition between the two lives.\nThe Taj is also seen
as a feminine architectural form, and is thought to embody Mumtaz Mahal herself.'
```

```
res=summarizer(txt)
res
```

```
'Due to the global attention that it has received and the millions of visitors it attracts, the Taj Mahal has become a prominen
t image that is associated with India, and in this way has become a symbol of India itself.[49]\n\nAlong with being a renowned
symbol of love, the Taj Mahal is also a symbol of Shah Jahan's wealth and power, and the fact that the empire had prospered und
er his rule.[50] Bilateral symmetry dominated by a central axis has been used by rulers as a symbol of a ruling force that brin
gs balance and harmony, and Shah Jahan applied that concept in the making of the Taj Mahal.[51] This makes the Taj a symbolic h
orizon.[52]\n\nThe planning and structure of the Taj Mahal, from the building itself to the gardens and beyond, is symbolic of
Mumtaz Mahal's mansion in the garden of Paradise.[51]'
```

```
from rouge import Rouge
r = Rouge()
r.get_scores(res, txt)
```

```
[{'rouge-1': {'r': 0.3739130434782609, 'p': 1.0, 'f': 0.5443037935066496},
  'rouge-2': {'r': 0.33163265306122447,
   'p': 0.9923664122137404,
   'f': 0.4971319274157247},
  'rouge-l': {'r': 0.3739130434782609, 'p': 1.0, 'f': 0.5443037935066496}}]
```

7. Advantages:

- Text Summarization decreases reading time, speeds up the research process, and expands the quantity of information that may fit in a given space
- Increased productivity
- Instantly effective
- Efficient for indexing
- Automatic summarization algorithms are less biased than human summarizes
- Doesn't miss out important facts

8. Disadvantages:

- Lack of fluency and coherence sometimes
- Sometimes it may misinterpret the author's original idea
- Difficult for evaluation

9. Applications:

- Search Engine Optimization
- Media
- Medical Cases
- Literature
- Search Marketing

10. Conclusion:

- The difficulty occurs in producing accurate summaries with suitable semantics, which include all aspects of text summarising, image summarization, article summaries, and multiple document summaries.
- We can also implement additional translation and text-to-speech capabilities to the model, making it more robust, adaptable and user-friendly.
- Summarization procedures must create a persuasive summary in a short amount of time, with little repetition and grammatically sound phrases.

11. Future scope:

Summarizing methods are all extractive, although the community is rapidly moving toward abstract summarization. An abstractive summarising can be created by using phrase compression and textual entailment techniques, even though a complete abstractive summarization would require a better grasp of natural language. Textual entailment aids in the detection of condensed versions of longer texts that convey the same message. We can construct shorter and more concise summaries using textual entailment.

12. Bibliography:

- https://medium.com/analytics-vidhya/simple-text-summarization-using-nltk-eedc36ebaaf8#:~:text=Text%20summarization%20is%20the%20process,relevant%20information%20within%20the%20Text.
- https://machinelearningmastery.com/gentle-introduction-text-summarization/